Métodos

Titular: Lic. Claudio O. Biale

Hacer programas modulares y reutilizables es uno de los objetivos centrales de la ingeniería de software. Java proporciona distintas construcciones que ayudan a lograr este objetivo. Los métodos son una construcción de este tipo.

El encabezado de un método especifica:
- los modificadores,
- el tipo de valor devuelto,
- el nombre del método y
- los parámetros del método.

Un método puede devolver un valor. Se debe especificar el tipo de datos del valor que devuelve el método. Si el método no devuelve un valor, el tipo de datos retornado es *void*.

La lista de parámetros se refiere al tipo, orden y número de los parámetros de un método. El nombre del método y la lista de parámetros juntos constituyen la firma del método. Un método puede o no contener parámetros, por eso se dice que el uso de parámetros es opcional.

Para devolver un valor se debe usar en el método la palabra clave *return*.

La palabra clave *return* también se puede usar en un método *void* para terminar el método y volver al llamante del método. Esto es útil ocasionalmente para eludir el flujo normal de ejecución en un método.

Los argumentos que se pasan a un método deben tener el mismo número, tipo y orden como los parámetros definidos en la firma de método.

Cuando un programa llama a un método, el control del programa se transfiere al método llamado. Un método llamado devuelve el control al llamante cuando se ejecuta su instrucción de devolución (*return*) o cuando se alcanza el final del método.

Un método que devuelve un valor (valor de retorno distinto de *void*) también puede ser invocado como una sentencia en Java. En este caso, el llamante simplemente ignora el valor devuelto.

Un método puede estar sobrecargado. Esto significa que dos métodos pueden tener el mismo nombre, siempre y cuando las listas de parámetros de cada método difieran.

Una variable declarada en un método se denomina variable local. El ámbito de una variable local empieza desde su declaración y continúa hasta el final del bloque que contiene la variable. Una variable local debe ser declarada e inicializada antes de ser utilizada.

Métodos Útiles

Titular: Lic. Claudio O. Biale

Java proporciona muchos métodos útiles en la clase *Math* para realizar funciones matemáticas comunes.

Métodos Trigonométricos

Método	Descripción
sin(radianes)	Devuelve el seno trigonométrico de un ángulo en radianes.
cos(radianes)	Devuelve el coseno trigonométrico de un ángulo en radianes.
tan(radianes)	Devuelve la tangente trigonométrica de un ángulo en radianes.
toRadians(grado)	Devuelve el ángulo en radianes para el ángulo en grado.
toDegree(radianes)	Devuelve el ángulo en grados para el ángulo en radianes.
asin(a)	Devuelve el ángulo en radianes para el inverso del seno.
acos(a)	Devuelve el ángulo en radianes para el inverso del coseno.
atan(a)	Devuelve el ángulo en radianes para el inverso de la tangente.

Métodos exponenciales

Método	Descripción
exp(x)	Devuelve e elevado a la potencia de x (e^x).
log(x)	Devuelve el logaritmo natural de x ($ln(x) = log_e(x)$).
log10(x)	Devuelve el logaritmo base 10 de x ($log_{10}(x)$).
pow(a, b)	Devuelve a elevado a la potencia de b (a^b).
sqrt(x)	Devuelve la raíz cuadrada de x para x >= 0.

Métodos de Redondeo

Método	Descripción
ceil(x)	x se redondea a su entero más cercano. Este entero se devuelve como un valor doble.
floor(x)	x se redondea a su entero más cercano. Este entero se devuelve como un valor doble.
rint(x)	x se redondea a su entero más cercano. Este entero se devuelve como un valor doble.
round (x)	Devuelve (int) Math.floor(x + 0.5) si x es un float y devuelve (long) Math.floor(x + 0.5) si x es un doble.

Métodos min, max y abs

Los métodos *min* y *max* devuelven los números mínimo y máximo de dos números (*int*, *long*, *float* o *doble*).

Por ejemplo:

- Math.max (4.4, 5.0) devuelve 5.0
- Math.min (3, 2) devuelve 2.
- Math.max (2, 3) devuelve 3
- Math.max (2.5, 3) devuelve 4.0
- Math.min (2.5, 4.6) devuelve 2.5

El método *abs* devuelve el valor absoluto del número (*int*, *long*, *float* o *double*):

Por ejemplo:

- Math.abs (-2) devuelve 2
- Math.abs (-2.1) devuelve 2.1

Valores Aleatorios

Este método genera un valor doble aleatorio mayor o igual a 0.0 y menor que 1.0 (0 <= Math.random () <1.0).
Puede usarse para escribir una expresión simple para generar números aleatorios en cualquier rango.

Por ejemplo:

- (int) (Math.random() * 10): Devuelve un entero aleatorio entre 0 y 9.
- 50 + (int) (Math.random() * 50): Devuelve un entero aleatorio entre 50 y 99.

De forma general:

a + Math.random () * b	Devuelve un número aleatorio entre a y a + b. Excluyendo a + b.

Excepciones

Titular: Lic. Claudio O. Biale

Introducción

El manejo de excepciones permite que un programa se ocupe de situaciones excepcionales y continúe su ejecución normal.

Son un mecanismo de control de errores en tiempo de ejecución. Una forma de hacer que la aplicación continúe la ejecución si se produce un error

Por ejemplo: ¿Qué sucede si no se puede abrir un fichero? ¿Y si se realiza una división por cero entre enteros?

Hay que considerar que si se controlan todos los posibles errores directamente, el código se puede volver ilegible. Con las excepciones se separa el código de un método del código que controla los errores.

Tipos de Excepciones

Las excepciones en Java son objetos, que se crean cuando ocurre una situación anómala. El objetivo es que se lance la excepción para que otra parte del código las capture y las trate.

Una excepción en Java es una instancia de una clase derivada de *java.lang.Throwable*. Java proporciona una serie de clases de excepción predefinidas, como *Error*, *Exception*, *RuntimeException*, *ClassNotFoundException*, *NullPointerException* y *ArithmeticException*. También es posible definir una clase de excepción propia extendiendo de *Exception*.

Las excepciones que se derivan de *Error* suelen estar relacionadas con la máquina virtual y no se espera que se capturen ni se traten. Las Excepciones derivadas de *Exception* sí que deben ser tratadas, y en algunos casos es obligatorio hacerlo para que el programa compile. Por otro lado no es necesario tratar las Excepciones que derivan de *RuntimeException*.

A las excepciones que derivan de *RuntimeException* y *Error* se las denomina excepciones no comprobadas *(unchecked)*. A las excepciones que derivan de *Exception* se las denomina excepciones comprobadas *(checked)*.

Capturar una Excepción

El código que puede generar la excepción debe encerrarse dentro de un bloque *try*, a continuación, la excepción se captura con un bloque *catch*.

```
try {
      // Código que puede generar la excepción
} catch (Exception e) {
      // Código para tratar el error
}
```

Cuando se produce una excepción dentro del bloque *try* se sale inmediatamente del bloque, si el bloque tiene asociada una cláusula *catch* adecuada para el tipo de excepción generado, se ejecuta el cuerpo de la cláusula *catch*.

Si el código dentro del bloque *try* puede generar más de una excepción, es posible capturar todas ellas:

```
try {
      //Código que puede provocar el error
} catch(IOException ioe) {
      //Código para tratar IOException
} catch(Exception e) {
      //Código para tratar Exception
}
```

Vale consignar que es posible capturar una excepción utilizando un tipo de excepción más general como por ejemplo *Exception*. Esto se puede hacer debido a que unas excepciones heredan de otras. Sólo se puede hacer con excepciones dentro de la misma jerarquía. Permite tratar de manera común varios tipos de excepciones distintos.

Hay que tener en cuenta que si un bloque de código lanza varias excepciones y se usan varios *catch*, la excepción se captura en el primer *catch* que se ajusta a la excepción. Los *catch* deben capturar las excepciones más concretas en primer lugar, y las más generales al final, si no se hace así, van a existir bloques *catch* en los que no se entrará nunca.

Por ejemplo considere los siguientes bloques de código:

<table>
<tr>
<td><pre><code>try {
      // código que genera
      // excepciones
} catch(IOException ioe) {
      // catch accesible
} catch (Exception e) {
      // catch accesible
}</code></pre></td>
<td><pre><code>try {
      // código que genera
      // excepciones
} catch(Exception e) {
      // catch accesible
} catch (IOException ioe) {
      // catch no accesible
}</code></pre></td>
</tr>
</table>

En el primero caso vamos de una excepción más concreta *(IOException)* hacia una excepción más general *(Exception)*. En el segundo caso vamos de una excepción más general *(Exception)* hacia una excepción más concreta *(IOException)*, en este caso al generarse una excepción se va tratar en el primer *catch* y nunca se va a acceder al segundo *catch*. Por ello es importante saber la jerarquía de clases de excepciones.

Varias clases de excepción se pueden derivar de una superclase común. Si un bloque *catch* captura los objetos de excepción de una superclase, también puede capturar todos los objetos de excepción de las subclases de esa superclase.

Cláusula finally

A veces, cuando se produce una excepción, la aplicación queda es un estado inestable. Al tratamiento de una excepción se le puede añadir al final un bloque *finally* que se

ejecuta siempre, se produzcan o no excepciones. Este bloque puede ser usado para cerrar ficheros, liberar recursos, etc.

A continuación se detalla un ejemplo de bloque *try-catch-finally*:

```
try {
      //código que genera excepciones
} catch(Exception e) {
      //tratamiento de la excepción
} finally {
      //código que se ejecuta siempre
}
```

Modelo de Excepciones

El modelo de gestión de excepciones de Java se basa en tres operaciones: declarar una excepción, lanzar una excepción y capturar *(catch)* una excepción.

La declaración/propagación de una excepción se especifica mediante la palabra reservada *throws*. El lanzamiento de una excepción se especifica mediante la palabra reserva *throw*.

Si una excepción no se captura se propaga hacia el método llamante, para que éste la trate, si no la trata, se sigue propagando hasta llegar al *main*, si en el *main* tampoco se trata, se aborta la ejecución del programa.

Si un método puede propagar una excepción se debe utilizar la palabra reservada *throws* en la cabecera del método, por ejemplo:

```
public void miMetodo() throws ArithmeticException
```

Si el método puede propagar varias excepciones se especifican las mismas en la cabecera del método separando las excepciones con una coma, por ejemplo:

```
public void miMetodo() throws excep1, excep2, ... , excepN
```

Si un programa detecta un error puede crear una instancia del tipo de excepción detectada y lanzarla, para ello debe usar la palabra reservada *throw*, por ejemplo:

```
throw new NullPointerException();
```

Cuando se lanza una excepción se sale inmediatamente del bloque de código actual, si el bloque tiene asociada una cláusula *catch* adecuada para el tipo de excepción generado, se ejecuta el cuerpo de la cláusula *catch*. Sino, se sale inmediatamente del bloque *(o método)* dentro del cual esta el bloque en el que se produjo la excepción y se busca una cláusula *catch* apropiada, este proceso continua hasta encontrar una cláusula *catch* adecuada. Si se llega a *main* y no se puede manejar entonces se genera un error.

Al definir un método, hay que decidir si las excepciones se propagan hacia arriba *(throws)* o si se capturan en el propio método *(catch)*.

Obtener Información de una Excepción

Un objeto de excepción contiene información valiosa acerca de la excepción. Se pueden utilizar los siguientes métodos de instancia de la clase *java.lang.Throwable* para obtener información sobre la excepción:

+getMessage() : String
+toString() : String
+printStackTrace() : void
+getStackTrace(): StackTraceElement[]

getMessage() : devuelve el mensaje que describe al objeto de excepción.

toString() : Devuelve la concatenación de tres cadenas de texto: (1) el nombre completo de la clase de excepción; (2) ":" (un colon y un espacio); (3) el valor retornado por el método *getMessage()*.

printStackTrace() : Imprime el objeto *Throwable* y la información de seguimiento de la pila de llamadas en la consola.

getStackTrace() : Retorna un arreglo de los elementos de la pila que representan el rastreo de pila perteneciente a este objeto de excepción.

Muti-catch

Desde JDK7 se permite utilizar la característica multi-catch para simplificar la codificación de las excepciones con el mismo código de manejo. La sintaxis es:

```
catch (Exception1 | Exception2 | ... | ExceptionN ex) {
     // Código que maneja la excepción
}
```

Nota: si un bloque *catch* maneja más de un tipo de excepción, entonces el parámetro *(en el ejemplo: ex)* es implícitamente *final*. Por ende no se le puede asignar ningún valor dentro del boque *catch*.

Conclusión

El manejo de excepciones separa el código de manejo de errores de las tareas normales de programación, facilitando así la lectura y la modificación de los programas.

El manejo de excepciones no debe usarse para reemplazar comprobaciones simples. Se debe realizar comprobaciones sencillas utilizando las sentencias *if* siempre que sea posible y reservar la gestión de excepciones para tratar situaciones que no se pueden manejar con sentencias *if*.

Objetos y Clases

Titular: Lic. Claudio O. Biale

Una clase es una plantilla para objetos. Define las propiedades de los objetos y proporciona constructores para crear objetos y métodos para manipularlos.

Una clase es también un tipo de datos. Puede usarse para declarar variables de referencia de un objeto. Una variable de referencia de un objeto contiene una referencia a ese objeto.

Un objeto es una instancia de una clase. Se utiliza el operador *new* para crear un objeto y el operador punto (.) para acceder a los miembros de ese objeto a través de su variable de referencia.

Un atributo o método de instancia se asocia con instancias individuales. Un atributo estático es compartido por todas las instancias de la misma clase. Un método estático es un método que se puede invocar sin usar instancias.

Todas las instancias de una clase pueden acceder a los atributos y métodos estáticos de la clase. Para mayor claridad, sin embargo, es mejor invocar atributos estáticos y métodos usando *NombreClase.atributo* y *NombreClase.metodo*.

Los modificadores de visibilidad especifican cómo se accede a una clase, un método y un dato (atributo). Se puede proporcionar un método getter (accessor) o un método setter (mutator) para permitir a los clientes ver o modificar los datos (atributos).

Un método getter tiene la siguiente firma *public tipoRetorno getNombreAtributo()*. Si el *tipoRetorno* es lógico, el método *get* debe definirse como *public Boolean isNombreAtributo()*. Un método setter tiene la firma *public void SetNombreAtributo (tipoDeDatos valor)*.

Todos los parámetros se pasan a los métodos usando paso por valor. Para un parámetro de tipo primitivo, se pasa el valor real; para un parámetro de tipo de referencia, se pasa la referencia al objeto.

La palabra clave *this* puede usarse para referirse al objeto llamante. También se puede usar dentro de un constructor para invocar a otro constructor de la misma clase.